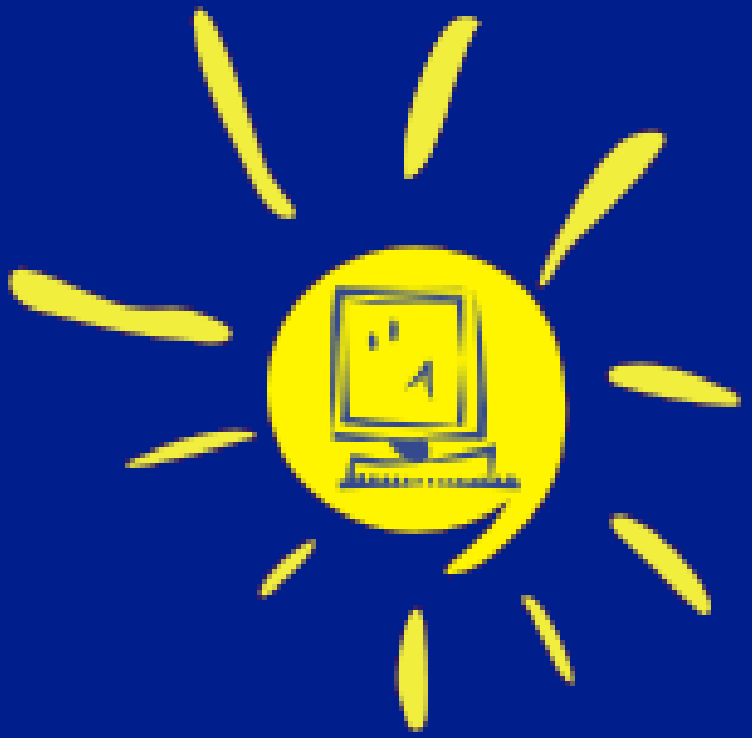


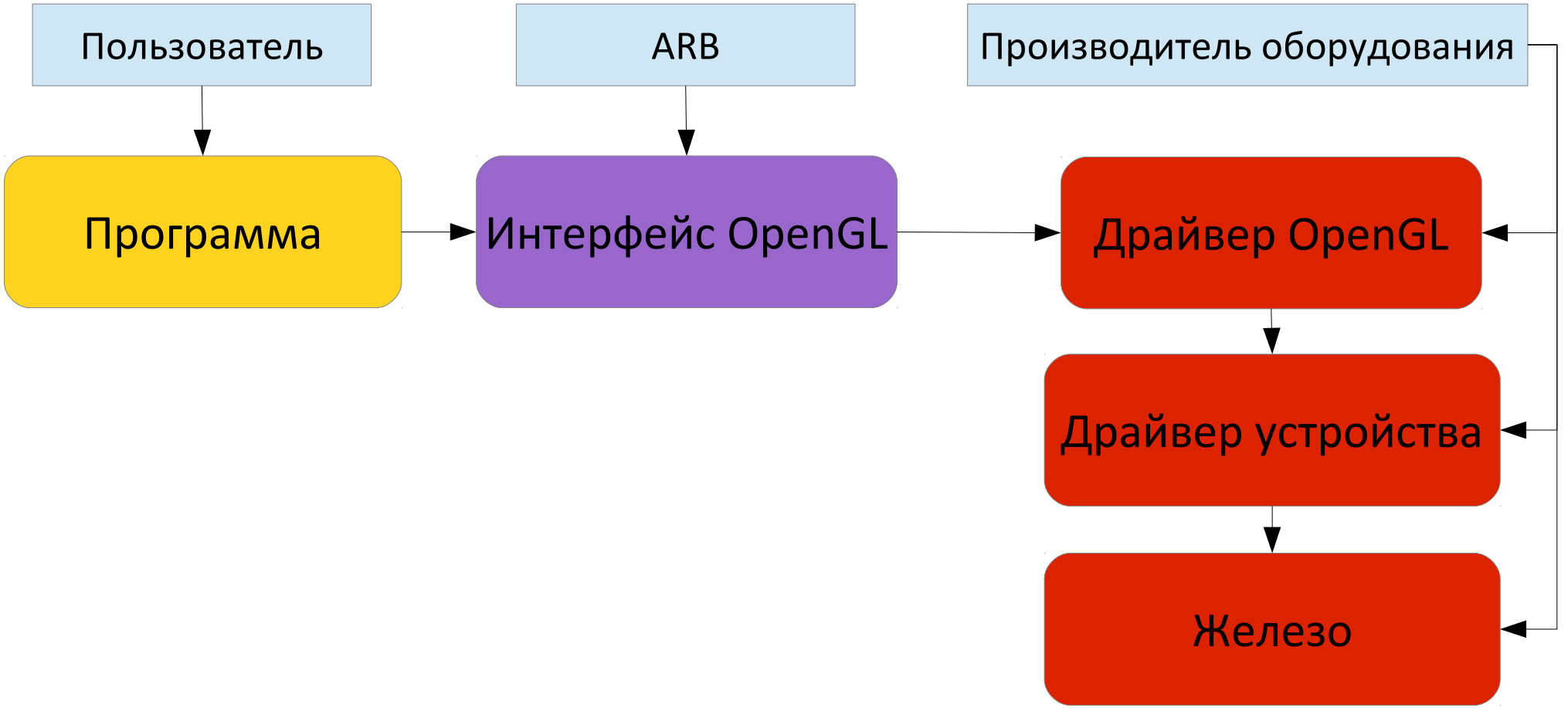
Графическая библиотека
OpenGL

Летняя компьютерная школа — 2012
12 августа

Графическая библиотека OpenGL

- История OpenGL начинается в 1992, когда была выпущена первая версия.
- OpenGL по своей сути не является графической библиотекой, а лишь **интерфейсом** к драйверу графического устройства.
- Стандартный интерфейс OpenGL рассчитан на язык C, однако существуют привязки и к другим языкам программирования. Например, C++, Java, Fortran, Perl, Python, Pascal, Ada и C#.

Графическая библиотека OpenGL

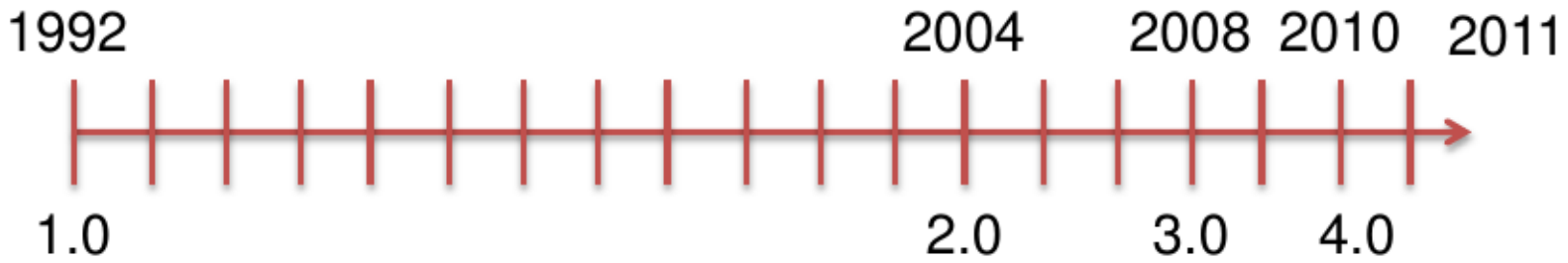


Графическая библиотека OpenGL

- В настоящее время стандарт библиотеки разрабатывается консорциумом ARB (OpenGL Architecture Review Board), в который входят все крупные производители графического оборудования.
- Последняя версия OpenGL 4.2 была выпущена 8 августа 2011.
- Каждый производитель вправе вносить свои нестандартные расширения для дополнительных функций, поддерживаемых только его оборудованием.
- Некоторые нестандартные расширения со временем могут войти в очередной стандарт.

Графическая библиотека OpenGL

За 20 лет существования OpenGL было выпущено 4 основных версии.



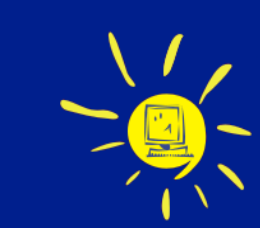
Microsoft DirectX был выпущен в 1995 году :)

В OpenGL 3.0 был введен новый API, который сильно отличается от OpenGL API предыдущих версий.

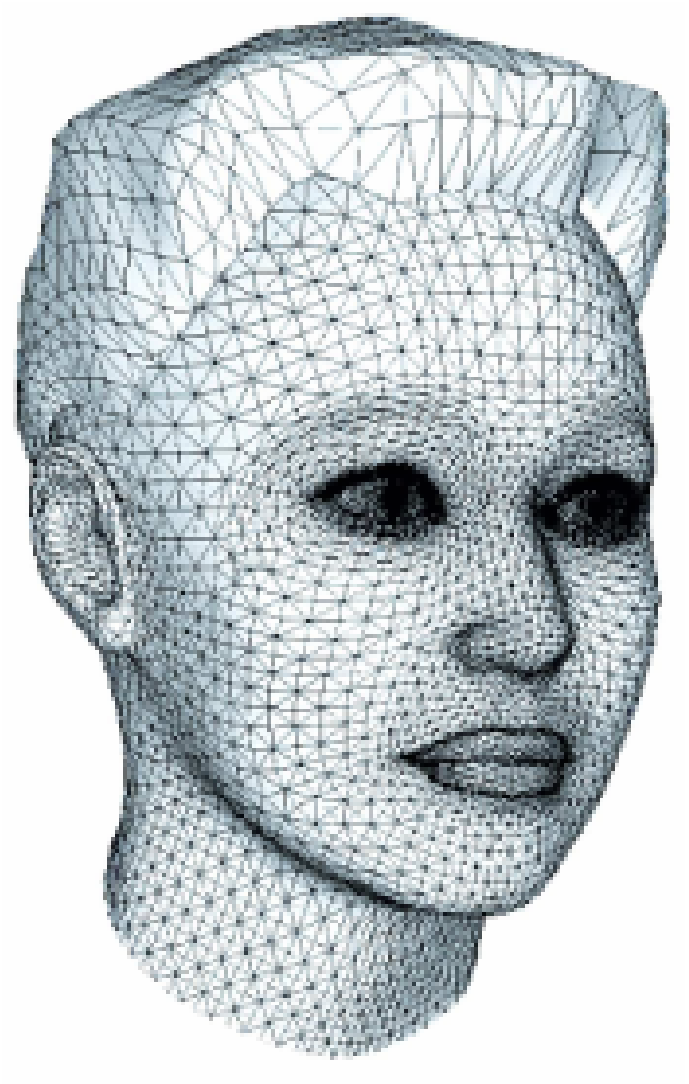
Графическая библиотека OpenGL

- В компьютерной графике используются различные модели для описания формы объектов.

- Одной из наиболее популярных моделей является полигональная модель, в которой фигура представляется в виде многоугольников, связанных через общие вершины.

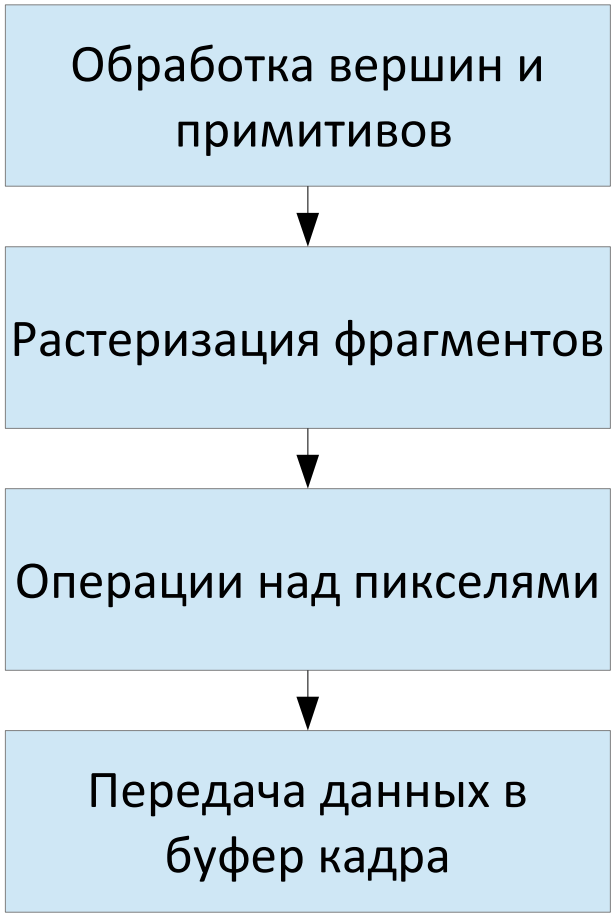


Графическая библиотека OpenGL



Графическая библиотека OpenGL

Как работает OpenGL?



Графическая библиотека OpenGL

- В OpenGL входят несколько составных частей. В частности, подбиблиотеки GLU и GLUT.

- Для подключения заголовочного файла, в котором находятся основные функции OpenGL в начале программы требуется добавить следующую строку:

```
#include <GL/gl.h>
```

- Аналогично, для подключения функций GLU и GLUT следует добавить

```
#include <GL/glu.h>
#include <GL/glut.h>
```

Графическая библиотека OpenGL

Синтаксис комманд OpenGL

```
type glCommandName[N][typeArg][v] (arg1, arg2, …, argN);
```

- gl — имя используемой библиотеки (например, glu или glut);
- CommanName — имя команды;
- N — количество аргументов;
- typeArg — тип аргументов (например, f для float);
- v — символ, указывающий, что в качестве параметров принимается не N аргументов, а указатель на массив из N аргументов.

Например,

- void glVertex3f(float x, float y, float z)
- void glVertex2dv(double *array)

Графическая библиотека OpenGL

Рисование в OpenGL состоит из трех основных шагов:

- очистка буферов;
- установка положения камеры;
- преобразование и рисование геометрических объектов.

Графическая библиотека OpenGL

- Очистка буферов производится с помощью функции glClear, которая в качестве аргумента принимает имена буферов, которые нужно очистить.

- В нашем примере функция glClear очищает буфер цвета (GL_COLOR_BUFFER_BIT).

- Перед выполнением очистки буфера выполняется функция glClearColor(R, G, B, A), которая устанавливает цвет, которым будет заполнен буфер.

Графическая библиотека OpenGL

- Атомарным графическим примитивом в OpenGL является точка (vertex).
- Из точек строятся все остальные примитивы.
- С каждой вершиной ассоциируются её атрибуты: положение в пространстве, цвет, вектор нормали и так далее.

Графическая библиотека OpenGL

- Положение вершины в пространстве

```
void glVertex[2 3 4][s i f d] (type coords)
void glVertex[2 3 4][s i f d]v (type *coords)
```

- Цвет вершины

```
void glColor[3 4][b s i f] (type coords)
void glColor[3 4][b s i f]v (type *coords)
```

- Нормаль

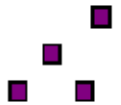
```
void glNormal3[b s i f d] (type coords)
void glNormal3[b s i f d]v (type *coords)
```

Графическая библиотека OpenGL

Операторные скобки объединяют точки в другие примитивы:

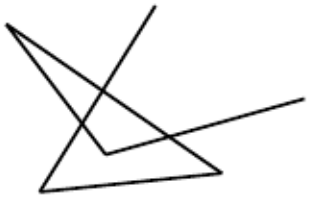
```
glBegin(mode);
  // описание точек
glEnd();
```

Графическая библиотека OpenGL



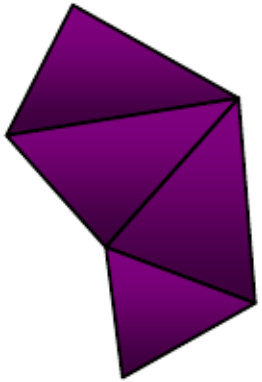
GL_POINTS

GL_LINES

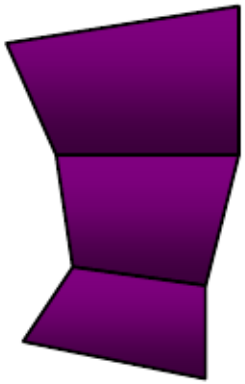


GL_LINE_STRIP

GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_QUAD_STRIPE

GL_POLIGON

GL_QUADS

Графическая библиотека OpenGL

- Для задания различных преобразований объектов в OpenGL используются линейные преобразования.

- Любое линейное преобразование в OpenGL задаётся матрицей чисел размером 4x4.

- Матрица может быть создана при помощи четырёх основных команд:

```
void glLoadIdentity()
void glTranslate[f d](x, y, z)
void glScale[f d](x, y, z)
void glRotate[f d](angle, x, y, z)
```

Графическая библиотека OpenGL

- Очень часто какие-то отдельные детали сцены оформляются в системе координат с центром в точке (0, 0, 0), а лишь затем переносятся на своё место матричными преобразованиями. Такие преобразования называются **модельными**.

- Для этого удобно использовать функции glPushMatrix и glPopMatrix:

```
glPushMatrix();
      glTranslate(...);
      glScale(...);
      glRotate(...);
      DrawSomething(); // наша функция рисования
glPopMatrix();
```

- Обратите внимание, что операции преобразования матрицы требуется писать в обратном порядке.

Графическая библиотека OpenGL

- После того, как сцена, состоящая из мелких деталей будет собрана в одно целое, обычно требуется применить **видовое** преобразование.

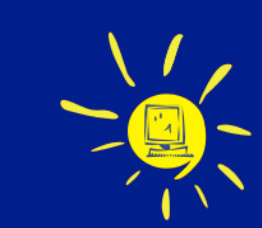
- Видовое преобразования заключается в том, что вся сцена трансформируется для того, чтобы камера смотрела на нее под специальным ракурсом.

- Для видового преобразования удобно использовать функцию

```
void gluLookAt(eye_x, eye_y, eye_z,
               cent_x, cent_y, cent_z,
               up_x, up_y, up_z)
```

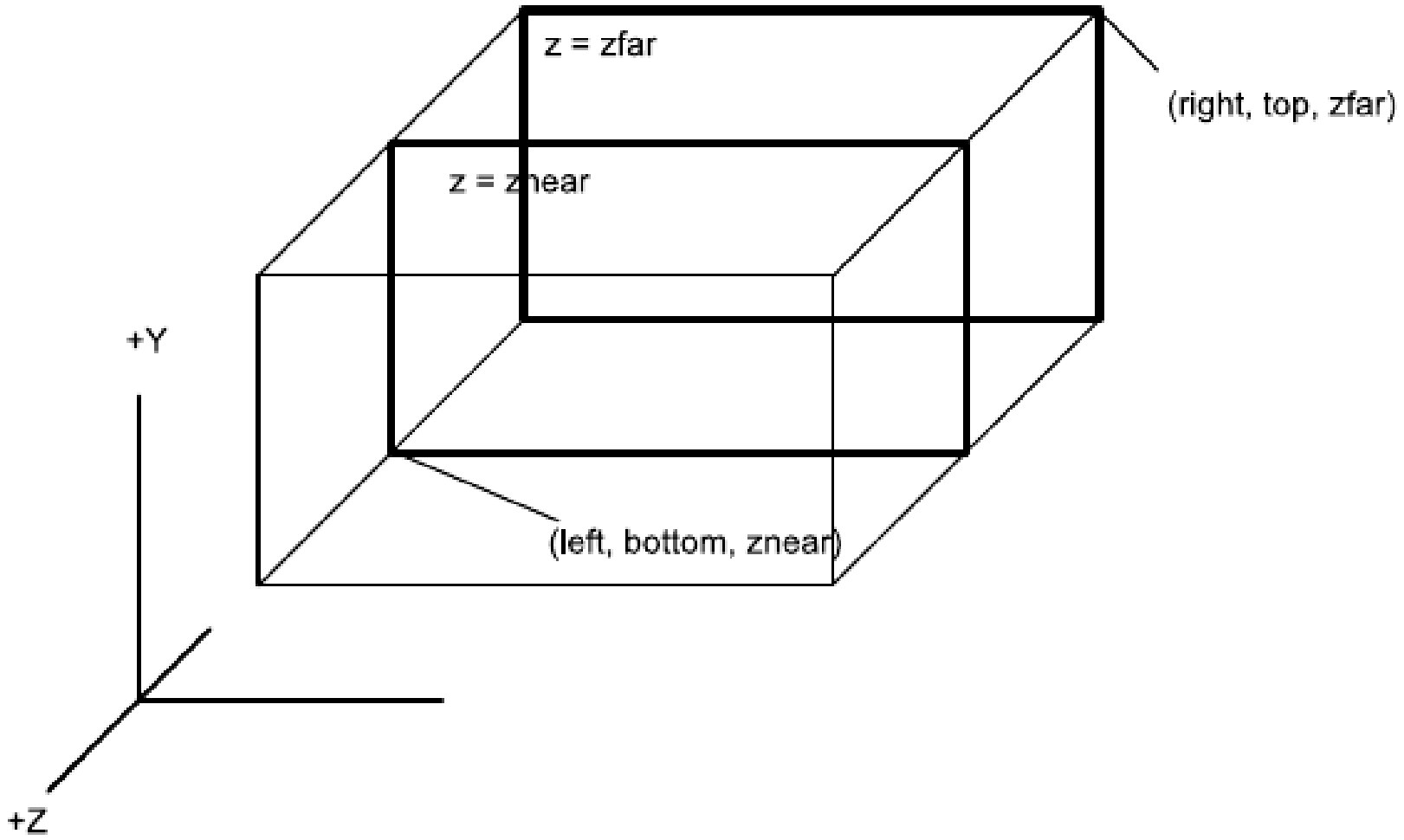
Графическая библиотека OpenGL

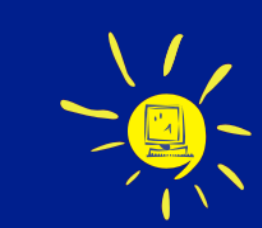
- Последнее преобразование — **проективное**. Оно требуется для преобразования трехмерной картинки в двумерную, то есть для переноса изображения сцены на монитор.



Графическая библиотека OpenGL

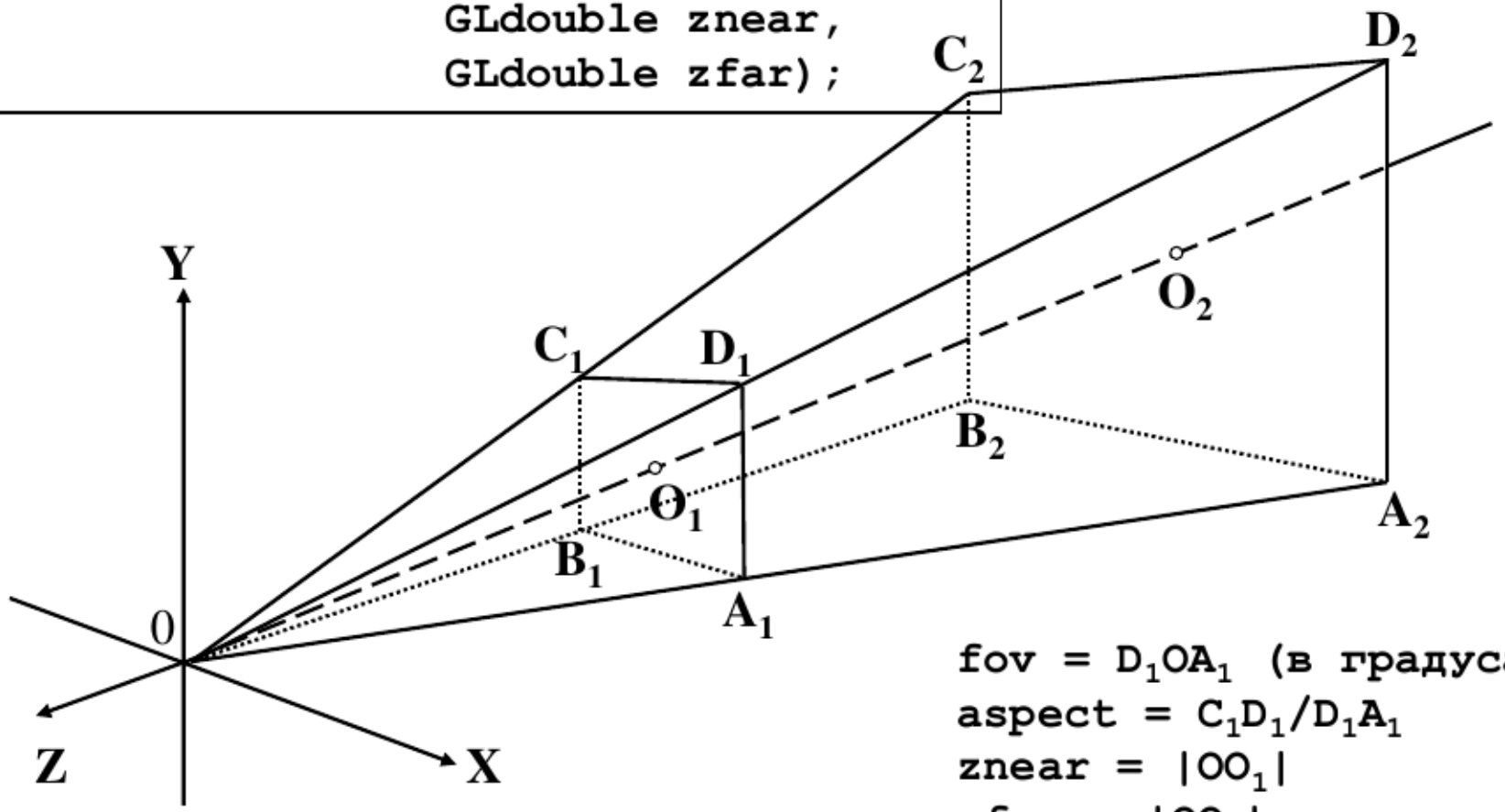
```
void glOrtho(left, right, bottom, top, near, far)
```





Графическая библиотека OpenGL

```
void gluPerspective(GLdouble fov,
                    GLdouble aspect,
                    GLdouble znear,
                    GLdouble zfar);
```



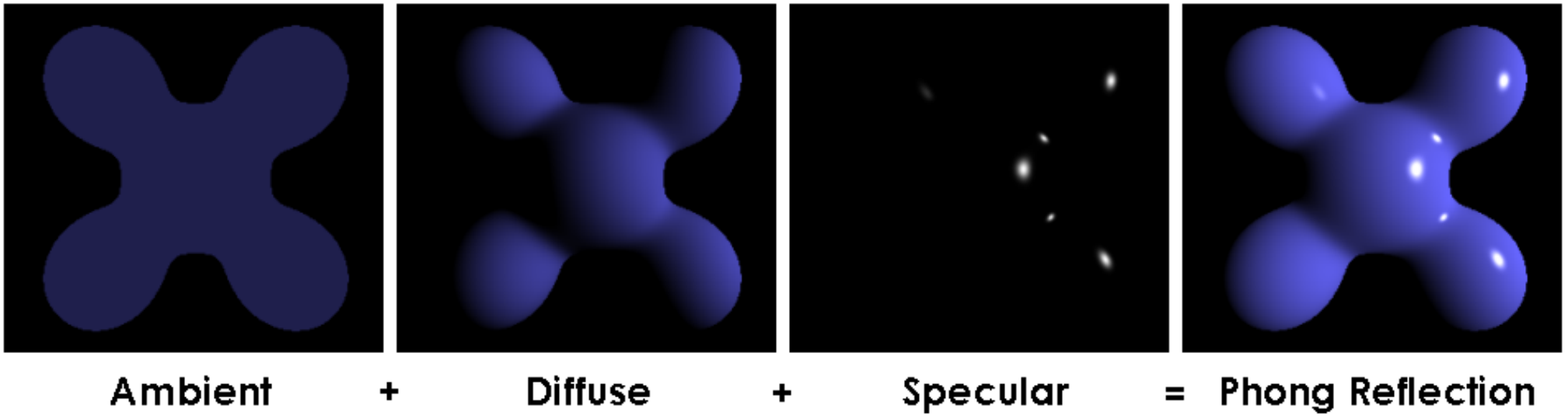
fov = D_1OA_1 (в градусах)
aspect = C_1D_1/D_1A_1
znear = $|OO_1|$
zfar = $|OO_2|$

Графическая библиотека OpenGL

- Одной из самых важных частей при построении сцены является освещение и материалы.

- В реальной жизни разные материалы по-разному отражают и поглощают свет. В OpenGL для имитации реальных поверхностей используется **модель освещения Фонга**.

Графическая библиотека OpenGL



Графическая библиотека OpenGL

- В OpenGL для задания типа материала используется функция:

```
void glMaterial[i f](face, pname, args...)
```

- Аргумент pname задаёт параметр материала:
  GL_AMBIENT — **рассеянный** цвет материала;
  GL_DIFFUSE — **диффузный** цвет материала;
  GL_SPECULAR — **зеркальный** цвет материала;
  GL_SHININESS — степень «блестящести» материала;
  GL_EMISSION — цвет излучаемого света.

- Аргумент face задаёт сторону примитива, к которой применяются параметры (GL_FRONT, GL_BACK, GL_FRONT_AND_BACK).

Графическая библиотека OpenGL

- Источник света задаётся командой

```
void glLight[i f](light, pname, args...)
```

- pname может принимать те же, значения, что и соответствующий аргумент в glMaterial.

- Дополнительно glLight может принимать аргумент pname равным GL_POSITION, которая принимает позицию источника света (координаты x, y, z, w). Если w=0, то источник света считается бесконечно удалённым.

Графическая библиотека OpenGL

- Источники света задаются константами `GL_LIGHT0`, `GL_LIGHT1`, `GL_LIGHT2`, …, которые передаются в функцию glLight, как аргументы.

- Для включения режима освещения требуется использовать команду

```
glEnable(GL_LIGHTING)
```

- Каждый источник света можно включить и выключить отдельно:

```
glEnable(GL_LIGHT0)
glDisable(GL_LIGHT0)
```

Графическая библиотека OpenGL

- Одним из наиболее используемых эффектов в компьютерной графике является эффект тумана, который позволяет придать реалистичности и скрыть некоторые огрехи реализации.

- Параметры тумана задаются с помощью функции

```
void glFog[i f] (pname, args).
```

Графическая библиотека OpenGL

Основным параметром (pname) является параметр GL_FOG_MODE, который определяет формулу, по которой вычисляется интенсивность тумана в точке. Возможные значения:

- GL_EXP;
- GL_EXP2;
- GL_LINEAR.

Кроме того, используются параметры:

- GL_FOG_DENSITY — «глубина» тумана;
- GL_FOG_COLOR — цвет тумана.

Текстуры и текстурные координаты

- Очень часто для придания объекту вида какого-либо материала (например, мрамора или дерева) приходится использовать текстуры.

- Для простоты каждой вершине примитива сопоставляется некоторая координата на текстуре с использованием функции

```
void glTextCoord2[s i f d] (x, y)
```


(0; 1)
(1; 1)
Текстура
(0; 0)
(1; 0)

Графическая библиотека OpenGL

Для загрузки изображения можно использовать функцию из библиотеки AUX:

```
void* auxDIBImageLoad(const char* filename)
```

Данная функция загружает текстуру из файла в формате *.bmp или *.dib и загружает данные во внутреннюю память.

После загрузки в память требуется передать изображение на видеокарту. Обычно для этого используют команду

```
void gluBuild2DMipmaps(target, components, width,
                       height, format, type, data)
```

Графическая библиотека OpenGL

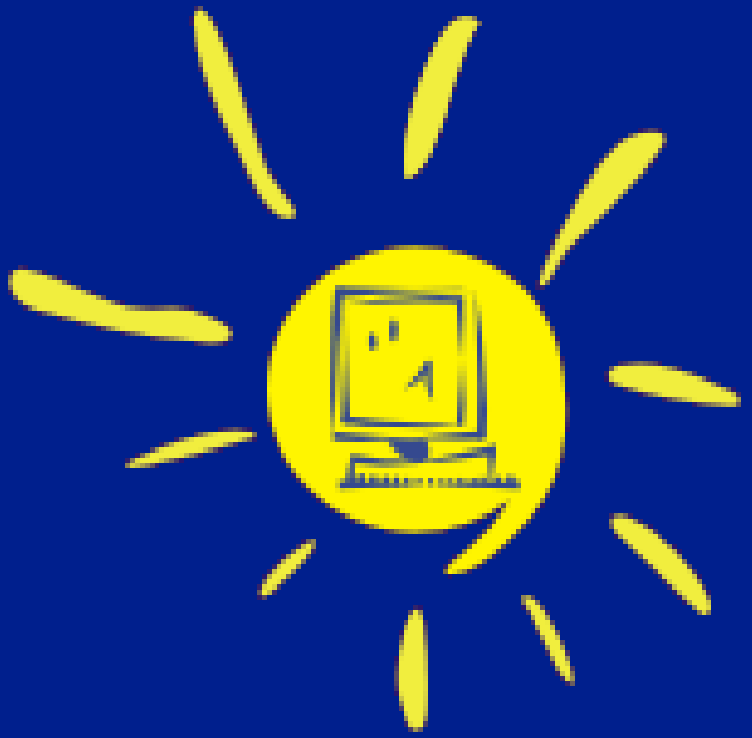
Пример загрузки текстуры:

```
glGenTextures(TexNumer, &TexFaceID);
glBindTexture(GL_TEXTURE_2D, TexFaceID[k]);
pImage = dibImageLoad(«texture.bmp»);
gluBuildMipmaps(GL_TEXTURE_2D, GL_RGB,
                pImage->sizeX,
                pImage->sizeY,
                GL_RGB,
                GL_UNSIGNED_BYTE,
                pImage->data);
```

Итог

Мы научились:

- строить из вершин более сложные объекты;
- использовать линейные преобразования при создании сцены;
- использовать различные проекции;
- правильно обрабатывать случаи, когда объекты находятся на переднем и заднем плане;
- задавать параметры материала и освещение;
- использовать эффект тумана;
- (возможно?) накладывать текстуры на примитивы.



Спасибо за внимание!